Application du contrôle pour garantir la performance des systèmes Big Data

Mihaly Berekmeri, Damián Serrano, Sara Bouchenak, Nicolas Marchand, Bogdan Robu

ComPAS'2014 : Parallélisme / Architecture / Système
Neuchâtel, Suisse, du 23 au 25 avril 2014

Application du contrôle pour garantir la performance des systèmes Big Data

M. Berekmeri, D. Serrano, S. Bouchenak, N. Marchand,B. Robu

GIPSA-lab, BP46 38402 Grenoble, France
email : {mihaly.berekmeri, nicolas.marchand, bogdan.robu}@gipsa-lab.fr

Distributed Computer Systems Group, LIG, BP53 38402 Grenoble, France
email : {damian.serrano, sara.bouchenak}@imag.fr

CNRS, France

Univ. Grenoble Alpes, F-38402 Grenoble, France

Résumé
Nous sommes à l'aube d'une énorme explosion de données et la quantité à traiter par les entreprises est de plus en plus grande. Pour faire face à ce chalenge, Google a développé MapReduce, un modèle de programmation parallèle qui est en train de devenir l'outil *de facto* pour l'analyse des systèmes Big Data. Bien que dans une certaine mesure son utilisation est déjà très répandue dans l'industrie, garantir les performances d'un système aussi complexe pose de grands problèmes et sa gestion nécessite un haut niveau d'expertise. Cet article répond à ces défis en proposant le premier système autonome qui garantit des contraintes de temps de réponse pour une charge de travail MapReduce simultanée. Nous développons le premier modèle dynamique d'une grappe MapReduce. De plus, un contrôle en boucle fermée est conçu et implémenté pour garantir un temps de réponse donné. Un contrôle d'anticipation de type "feedforward" est également rajouté pour améliorer la réponse du système en présence de perturbations, en l'occurrence, la variation du nombre de clients. L'approche est validée en ligne sur une grappe MapReduce avec 40 nœuds utilisant une charge de travail intensive de type Business Intelligence. Nos expériences montrent que le contrôle ainsi conçu peut garantir les contraintes de temps de réponse.

Mots-clés : réjection de perturbation, contrôle pour ordinateurs, Cloud Computing, Big Data

---

1. Introduction

Alors que nous entrons dans l'ère de Big Data (le mot Big Data est utilisé pour désigner un ensemble de données tellement volumineux qu'il est très difficile à analyser et à manipuler avec des outils classiques de gestion de base de données ou de gestion de l'information), l'augmentation très rapide de la quantité des données produites apporte de nouveaux défis dans l'analyse et le stockage de ces données. Récemment, il y a un intérêt croissant dans des domaines clés tels que l'extraction de données en temps réel, qui révèlent un besoin urgent de traitement des données très volumineuses sous contraintes strictes de performance. Ces applications peuvent aller de la personnalisation en temps réel des services internet, l'aide à la décision pour des analyses financières rapides, aux contrôleurs de trafic. La forte augmentation de la quantité de données non structurées disponibles appelle donc à un changement de perspective de l'approche traditionnelle des bases de

données, en allant vers une plateforme efficace de calcul distribué conçue pour la manipulation de pétaoctets d'information. Ceci impose l'adaptation des fournisseurs de services internet à des implémentations sur des plateformes informatiques distribuées et une façon d'y parvenir est d'adopter le modèle de programmation populaire appelé MapReduce. Son succès réside dans sa simplicité d'utilisation, sa extensibilité et tolérance aux fautes. MapReduce est soutenu et intensivement utilisé par les plus grands leaders de l'industrie informatique tels que Google, Yahoo, Facebook et Amazon. Par example, Google exécute plus de 100.000 tâches MapReduce par jour, Yahoo a plus de 40.000 ordinateurs qui exécutent des tâches MapReduce et Facebook l'utilise pour analyser plus de 15 pétaoctets de données. Le modèle de programmation MapReduce a été initialement développé par Google en 2008 comme un algorithme de calcul parallèle qui vise à gérer automatiquement le partitionnement des données, la cohérence et la réplication, ainsi que la répartition des tâches, la planification, l'équilibrage de charge et la tolérance aux fautes (voir [5] pour plus de détails).
Dans le même temps, nous observons un intérêt grandissant des informaticiens pour la théorie de la commande, qui permet de gérer automatiquement les configurations des systèmes informatiques complexes. Des publications récentes sur le contrôle des systèmes informatiques en temps continu montrent l'émergence de ce nouveau domaine de recherche. Nous pouvons par exemple citer [15], où la théorie de stabilité de Lyapunov a été utilisé pour contrôler des serveurs de base de données, de même nous pouvons citer [16] qui a utilisé une approche de type "boîte noire" pour contrôler des systèmes de services Web ou [9] pour contrôler des serveurs HTTP. Il faut aussi souligner l'émergence du contrôle des systèmes informatiques en utilisant les systèmes à événements discrets, voir [18] pour une vue d'ensemble. De plus, pour voir les approches différentes de l'utilisation de la théorie du contrôle pour les systèmes informatiques voir [12], [1] and [6].
L'objectif de ce papier est d'utiliser la théorie de la commande pour garantir la performance de systèmes MapReduce. MapReduce est un paradigme de programmation développé pour les calculs parallèles et largement utilisé par des applications réparties, voir [5]. Bien que MapReduce cache aux utilisateurs la plus grande partie de la complexité liée au parallélisme [1], le déploiement d'une implémentation efficace MapReduce nécessite toujours un haut niveau d'expertise. C'est par exemple le cas lors du réglage des paramètres de la configuration de MapReduce comme présenté dans [25, 24, 10] ou pour garantir des objectifs de performance comme indiqué dans [26, 23]. Par objectif de performance, on entend généralement le temps de réponse, c'est à dire le temps nécessaire au programme en cours d'exécution sur le cloud pour répondre à une requête du client. Pour exécuter un travail MapReduce, l'utilisateur doit fournir au moins trois éléments : les données à traiter, une fonction Map, et une fonction Reduce. Du point de vue de la théorie des systèmes, les deux fonctions Map et Reduce peuvent être considérées seulement comme des modèles boîte noire, car elles sont spécifiques à l'application. De plus, nous ne supposons aucune connaissance *a priori* sur leur comportement. Sans profilage, aucune hypothèse ne peut être faite concernant leur exécution, leur utilisation des ressources ou la quantité de données de sortie qu'ils produisent. En plus de cela, de nombreux facteurs (indépendantes des les données d'entrée et de la fonctions Map et Reduce) influencent les performances des tâches MapReduce : CPU, entrées / sorties et le biais du réseau [20], les défaillances matérielles et logicielles [19], l'hypothèse d'homogénéité des nœuds Hadoop qui n'est pas réaliste (Hadoop, voir [25], est l'implémentation open source la plus utilisée de MapReduce) [29, 17], et le changement très rapide et aléatoire de la charge de travail [4]. Tous ces facteurs sont vus par le système MapReduce comme des perturbations.
Compte tenu de tous ces défis les contributions de cet article sont :
– création du premier modèle dynamique d'un système de MapReduce.
– la conception et l'implémentation du premier correcteur en ligne capable de garantir des contraintes de temps de réponse pour une charge de travail MapReduce simultanée et aléatoire.
La suite de l'article est organisée comme suit. La Section 2 présente la modélisation du système

1. Par les utilisateurs MapReduce, nous entendons les entreprises souhaitant utiliser MapReduce pour leurs propres applications, généralement les fournisseurs de services Internet.

MapReduce et l'environnement d'évaluation. La Section 3 décrit les correcteurs proposés pour garantir des contraintes en terme de temps de réponse. La Section 4 présente l'état de l'art et la Section 5 décrit nos conclusions.

2. Modélisation du système MapReduce

2.1. Difficultés de modélisation

Les défis de modélisation d'un tel système sont très nombreux mais les deux principaux sont :

1. MapReduce a une architecture complexe, ce que implique une grande difficulté dans la modélisation détaillé de son comportement. Un tel modèle serait par exemple un modèle hybride non-linéaire de très grande dimension, variant dans le temps, et par conséquent vraisemblablement d'une valeur discutable. En outre, étant donné qu'il est très difficile d'intégrer l'effet de points d'engorgement - réseau, IO, CPU - dans un modèle mathématique, plusieurs hypothèses fortes doivent être faites (comme celle qu'un seul travail est en cours d'exécution à chaque moment dans la grappe) qui n'est pas valable dans les grappes réelles. De plus, nous devons mentionner que la performance des systèmes MapReduce varie d'une distribution à l'autre et même dans le cadre d'une même distribution, en raison du développement continu. Cela complique en plus la construction d'un modèle général.
2. Les modèles des systèmes MapReduce existants ne prennent en compte généralement que la réponse statique, par conséquent ils sont très difficiles à exploiter. L'une des principales raisons de l'absence de modèles dynamiques est la grande complexité du système mentionné précédemment.

2.2. Aperçu de la modélisation

Dans cette section, nous abordons les défis présentés précédemment.

Gérer la complexité du système. On observe que, même si le système est non linéaire, nous pouvons linéariser autour d'un point de fonctionnement caractérisé par un nombre de référence de nœuds et de clients. Après que le fournisseur de service ait décidé le nombre de nœuds qu'il souhaite pour ses requêtes (généralement basé sur des contraintes financières), notre algorithme augmente progressivement le nombre de clients qu'il peut accepter, jusqu'à ce que le débit de la grappe soit maximisé (ce qui est très important pour des raisons environnementales et financières). Ce point, où la grappe est entièrement utilisée mais sans être saturée, sera le point de fonctionnement pour la linéarisation. Il s'agit du mode de fonctionnement optimal.

Capturer la dynamique du système. L'un des défis importants dans les déploiements actuels de MapReduce est qu'il doit assurer certains temps de réponse minimum. Par conséquent, notre objectif de contrôle est de garder la moyenne des temps de service inférieure à un seuil donné, pour les travaux ayant fini dans la dernière fenêtre de temps. Cette fenêtre temporelle est introduite pour attribuer une dynamique mesurables au système. La définition de la longueur de la fenêtre n'est pas très simple car plus la fenêtre est grande plus nous perdons la dynamique tandis que plus elle est petite plus les mesures son bruitées. La valeur optimale du choix de la taille de la fenêtre est au-delà de la portée de cet article et pour l'instant nous avons choisi la taille de la fenêtre pour être au moins deux fois plus grande que le temps d'exécution moyen de la tâche.

Le choix des entrées de commande parmi les nombreux paramètres de Hadoop (plus de 170) n'est pas non plus évident. Comme nous voulons que notre modèle ne soit pas dépendant de la mise en œuvre, nous ne prenons en considération que les paramètres ayant une forte influence indépendamment de la version MapReduce utilisée. Deux de ces facteurs, qui ont été identifiés comme étant parmi ceux ayant le plus grand degré d'influence sont le nombre de Mappers et de Reducers disponibles pour le système, voir [28]. Comme ces paramètres sont fixes par nœud, nous avons choisi le nombre de nœuds comme notre entrée de commande car il influence les deux à la fois.

2.3. Structure du modèle proposé

La grande complexité d'un système MapReduce et les changements continus de son comportement, en raison de mises à niveau et d'améliorations logicielles, nous a incité à éviter l'utilisation d'une technique de modélisation boîte blanche et d'opter pour une technique qui est agnostique à ces derniers. Cela nous amène à une technique de modélisation boîte grise ou boîte noire. La ligne de démarcation entre ces deux techniques n'est pas bien définie, mais nous considérons notre modèle comme un modèle boîte grise puisque la structure du modèle a été définie sur la base de nos observations des régions de linéarité de fonctionnement du système.

Nous proposons un modèle dynamique qui prédit les performances d'une grappe MapReduce, dans notre cas le temps de réponse moyen, en fonction du nombre de nœuds et du nombre de clients. à notre connaissance, c'est le premier modèle dynamique orienté performances pour les système MapReduce.

La structure de notre modèle est représentée dans la Figure 1. Notre entrée de commande $u(k)$ est donnée par le nombre de nœuds dans la grappe tandis que les changements de clients $d(k)$ est considéré comme une perturbation mesurable. Notre sortie $y(k)$ est la moyenne des temps de réponse des tâches dans le $k^{ième}$ intervalle de temps. Comme autour du point de fonctionnement notre système est linéaire, nous pouvons appliquer le principe de superposition pour calculer la sortie :

$$y(k) = Z_C(z)d(k) + Z_N(z)u(k) \tag{1}$$

où Z_N est le modèle discret entre le temps de réponse et le nombre de nœuds et Z_C est le modèle discret entre le temps de réponse et le nombre de clients.

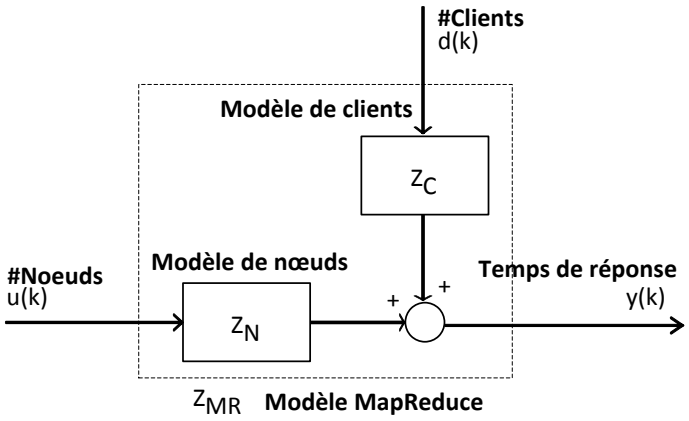

FIGURE 1 – Schéma du modèle théorique de contrôle MapReduce

2.4. Environnement d'évaluation.

La suite de tests MapReduce (MRBS) développé par [19] est une suite de tests pour les fiabilités des systèmes MapReduce. MRBS peut émuler plusieurs types de charges de travail et injecter différents types de défauts dans un système MapReduce. Les charges de travail émulés par MRBS sont destinés à couvrir cinq domaines d'application : systèmes de recommandation (RS), business intelligence (BI), bio-informatique, traitement de texte et analyse de données. Ces charges de travail ont été sélectionnés pour représenter un éventail de charges, du calcul intensif (charge processeur importante) à une charge de données intensive (par exemple, le BI). Une des forces de la suite de

tests MRBS est d'imiter les demandes des clients. Une demande peut être constitué d'un ou plusieurs *job* MapReduce. Ces *jobs* sont des exemples de ce que peut être une demande typique dans un déploiement réel d'un système de MapReduce.
Grid5000 est une infrastructure de grappe à l'échelle nationale française composée de 5000 CPUs. Il été mis au point pour faciliter la recherche de calcul parallèle. Il fournit un outil scientifique pour mener des expériences distribués à grande échelle, voir [2].
Toutes les expériences dans cet article ont été effectuées en ligne sur Grid5000 avec un seul grappe de 40 nœuds. La configuration de la grappe peut être vue dans le Tableau 1.

Cluster	**CPU**	**Mémoire**	**Stockage**	**Réseau**
40 nœuds Grid5000	4 cores/CPU Intel 2.53GHz	15GB	298GB	Infiniband 20G

TABLE 1 – Configuration matérielle des grappes MapReduce

Pour nos expériences, nous utilisons le cadre de la mise en œuvre de MapReduce open source Apache Hadoop v1.1.2 et l'outil d'évaluation de haut niveau MRBS. Une charge de travail intensif de données BI est sélectionné comme notre charge de travail. L'indice de référence BI se compose d'un système d'aide à la décision pour un fournisseur en gros (des entreprises qui achètent et/ou vendent exclusivement à d'autres entreprises professionnels). Chaque requête émule une demande du client sur une grande quantité de données (10 Gb dans notre cas). Pour générer la demande d'une client Apache Hive est déployé au-dessus de Hadoop. Il convertit les requêtes SQL dans une série d'emplois MapReduce.
Une version simplifiée de notre dispositif expérimental peut être vu dans la Figure 2

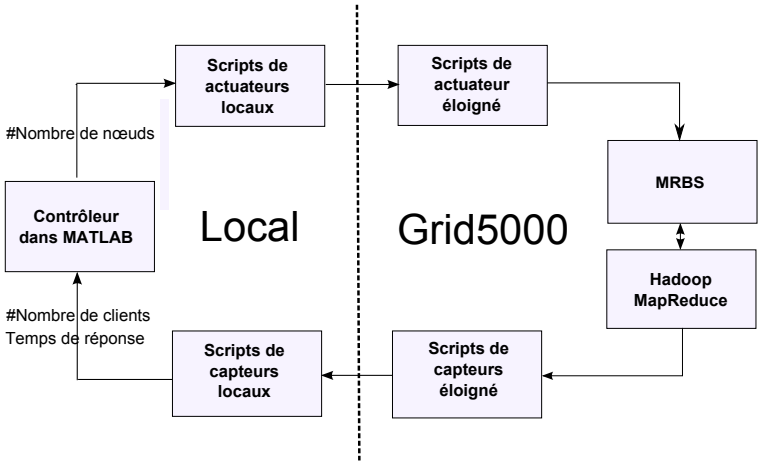

FIGURE 2 – Environnement d'évaluation

Le contrôle est implémenté en Matlab et toutes les mesures sont effectuées en temps réel en ligne. Nous mesurons le nombre de clients et le temps de réponse de la grappe et nous utilisons le nombre de nœuds pour assurer les contraintes de temps de réponse quel que soit le nombre fluctuant de clients. Tous nos capteurs et actionneurs sont mis en œuvre dans des scripts Linux Bash.

2.5. Identification des modèles

Les deux modèles ont été identifiés en utilisant la réponse indicielle. En automatique la réponse indicielle est la réponse d'un système à une augmentation de type échelon, ce qui permet de déduire les caractéristiques dynamique de le système. La linéarité observée autour du point de fonctionnement, l'absence de dépassement et la décroissance exponentielle, indiquent que, au moins dans une première approche, la réponse pourrait être modélisé avec un modèle linéaire de premier ordre

avec retard. Les paramètres du modèle sont identifiés en utilisant la méthode d'estimation prédiction d'erreur. Cette méthode présente l'avantage de mettre l'accent sur la précision du modèle pour prédire l'observation suivante plutôt que sur la différence avec un modèle statistique correspondant, comme pour les méthodes des moindres carrés et maximum de vraisemblance. En plus, il a été montré que cette méthode peut fournir des résultats optimaux (matrice de covariance minimale) dans le cas où la structure du modèle choisi reflète bien la vérité, voir [14]. Comme le système a des grandes constantes de temps (>300s) on déduit qu'une période d'échantillonnage de 30 secondes est suffisante. Les modèles sont identifiés comme des modèles en temps continu et nous utilisons la transformation bilinéaire Tustin pour les discrétiser.

2.5.1. Identification du système sans perturbation

Le modèle identifié correspondant aux changements de nœuds peut être vu dans la Figure 3. Une

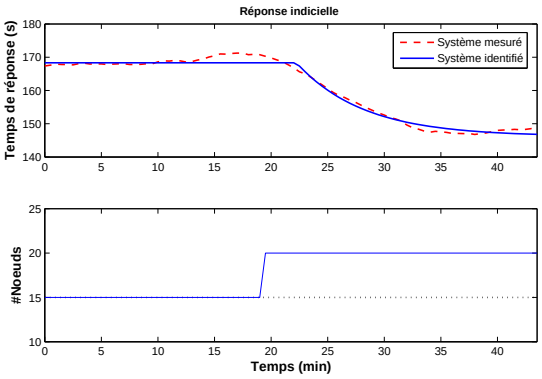

FIGURE 3 – Identification du système non perturbé. Il prédit l'effet de changements de nœuds sur le temps de réponse.

augmentation de type échelon est utilisée sur le nombre de nœuds pour identifier le modèle entre le temps de service et le nombre de nœuds. Le modèle suit la dynamique du système réel avec une précision de 89%. L'équation (2) présente la fonction de transfert en temps discret du système sans perturbations. La fonction de transfert est une représentation mathématique de la relation entre l'entrée et la sortie du système , le nombre de nœuds et le temps de réponse dans notre cas.

$$Z_N(z) = z^{-5}\frac{-0.17951(z+1)}{z-0.919} \tag{2}$$

2.5.2. Identification du modèle de perturbation

Figure 4 montre la réponse indicielle dans le cas d'une variation du nombre de clients.

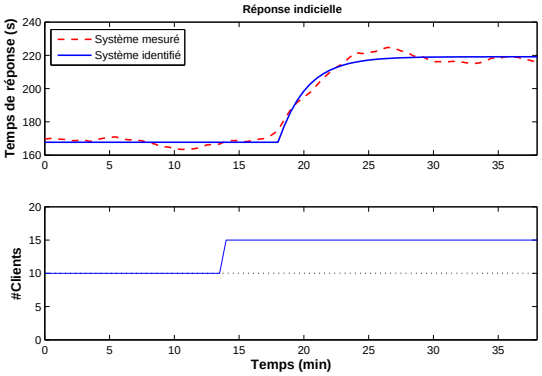

FIGURE 4 – L'identification du modèle de perturbation. Il saisit l'effet des changements dans le nombre de clients sur le temps d' exécution.

Comme nous pouvons voir, le modèle identifié colle bien au comportement du système réel, présentant un précision de 87.94%. L'équation (3) nous donne la fonction de transfert en temps discret du modèle de perturbation :

$$Z_C(z) = z^{-8}\frac{1.0716(z+1)}{z-0.7915} \tag{3}$$

Les deux fonctions de transfert précédemment identifiées sont stables, systèmes de premier ordre avec des pôles à l'intérieur du cercle unité. Par conséquent, le système total est intrinsèquement stable en boucle ouverte (sans contrôle).

3. Contrôler MapReduce

Défis de contrôle contrôler MapReduce présente plusieurs défis spécifiques. Le premier est dû au temps de latence assez important (>90s) consécutif à l'ajout de nœuds et de clients qui est une procédure complexe. D'autres défis sont spécifiques à la mise en œuvre. Par exemple, l'ajout et la suppression de nœuds implique toujours un coût énergétique et financier, il est donc nécessaire d'avoir une réponse du système en boucle fermée sans voire avec de très faibles dépassements. Un autre défi intéressant est lié à la quantification. Le nombre de nœuds que nous pouvons rajouter ou supprimer doit être un entier strictement positif. Enfin, comme la performance du système peut varier dans le temps en raison des nombreux points d'engorgement, l'algorithme de commande doit être assez robuste pour gérer les incertitudes de modélisation.

Architecture de contrôle Un correcteur proportionnel intégral (PI) est choisi car il est prouvé que pour notre système (c'est à dire un système de premier ordre avec retard - voir l'équation (2), ce type de correcteur est suffisant même si le système est complexe ([8]). En outre, un dispositif de commande de rétroaction à action intégrale offre plusieurs avantages bien connus : il corrige les erreurs qui découlent des imperfections de modèle et a des propriétés éprouvées de rejet de perturbations non mesurées et/ou non modélisés. Ce correcteur donne une commande proportionnelle à l'erreur est élimine l'erreur final en utilisant l'intégrale de l'erreur. Si nous savons le modèle dynamique du la système, il existe des méthodes éprouvées pour calculer les paramètres du correcteur.
De plus, puisque nous pouvons mesurer avec précision les perturbations, nous ajoutons également un correcteur d'action directe (feedforward) qui améliore le temps de réponse car il neutralise l'effet de la perturbation avant qu'elle n'influence la sortie. Le schéma complet de l'architecture de contrôle peut être vu dans la Figure 5. Les variables utilisées dans la figure sont définies dans le Tableau 2.

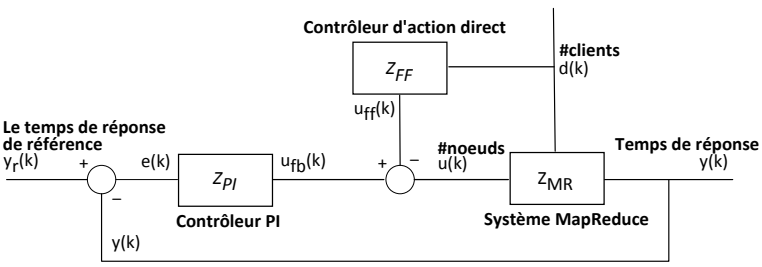

FIGURE 5 – Architecture de commande MapReduce

y_r	La référence du temps de réponse moyen défini dans le SLA.
y	Sortie du système - temps de réponse de CIs.
u	Entrée de commande du système - nombre de nœuds dans le système.
u_{fb}	L'entrée de commande du correcteur PI.
u_{ff}	L'entrée de commande du correcteur d'action directe feedforward.
e	Erreur, entrée du correcteur.
d	Entrée de perturbation - nombre de clients exécutant des travaux.
Z_{MR}	La modèle de temps discret de système de MapReduce.
Z_{PI}	Fonction de transfert discrétisée du PI.
Z_{FF}	Fonction de transfert discrétisée du feedforward.

TABLE 2 – Définition de variables de contrôle.

3.1. Etude de la boucle ouverte

Dans la Figure 6 nous observons l'effet d'une augmentation progressive de l'entrée exogène (le nombre de clients) sur le système sans correction. Une telle augmentation du nombre de clients se produit fréquemment dans la pratique, voir par example [11] qui analyse pendant 10 mois une grappe de calcul intensif de Yahoo.

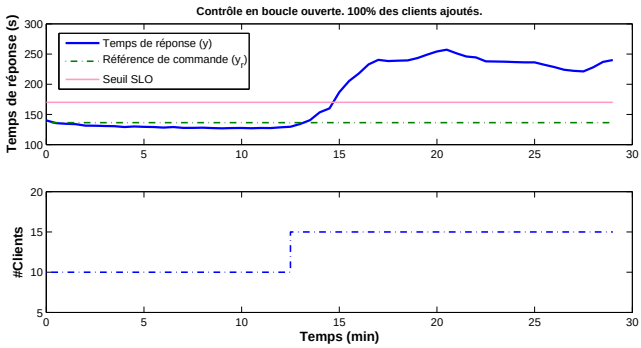

FIGURE 6 – MapReduce essai en boucle ouverte.

Dans notre cas, nous pouvons voir que pour une augmentation du nombre de clients, le temps de réponse de la grappe dépasse rapidement le seuil de référence défini à l'avance (SLA-Service Level Agreement). Afin de respecter le seuil fixé nous employons un correcteur PI.

3.2. Commande PI

La forme standard d'un correcteur échantillonné de type PI est donnée dans l'équation (4) :

$$u_{fb}(k) = u_{fb}(k-1) + (K_p + K_i)e(k) + K_i e(k-1) \tag{4}$$

Les paramètres K_p et K_i sont déterminés pour assurer la stabilité en boucle fermée (système commandé) et un dépassement de notre sorti de la consigne de 0%. Comme nous voulons minimiser le changement trop rapide du nombre de nœuds (en raison des contraintes financières et énergétiques), nous évitons un correcteur très agressif. Sur cette base nous calculons les valeurs de $K_p = 0.0012372$ et $K_i = 0.25584$.

Le résultat expérimental de la mise en œuvre du correcteur est donné dans la Figure 7 et montre la réponse du système avec correcteur à une variation de 100% du nombre de clients. Comme le correcteur est censé avoir un temps de stabilisation plus grand, le seuil SLA est dépassé pendant un court instant mais le correcteur ramène toujours le temps de réponse à la valeur de référence. Le correcteur augmente constamment le nombre de nœuds jusqu'au moment où le temps de réponse

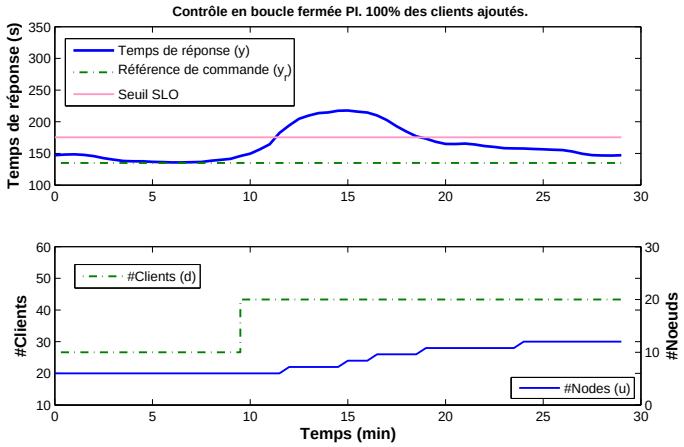

FIGURE 7 – Correcteur PI

est inférieur à la référence. Toutefois, afin d'éviter le non respect du SLA et pour améliorer le temps de réaction de la commande, un correcteur feedforward est ajouté.

3.3. Commande d'action directe feedforward avec PI

L'effet du correcteur d'action directe avec le correcteur PI déjà employé peut être vu dans la Figure 8. Le dispositif de commande d'anticipation est conçu pour rejeter pro-activement les pertur-

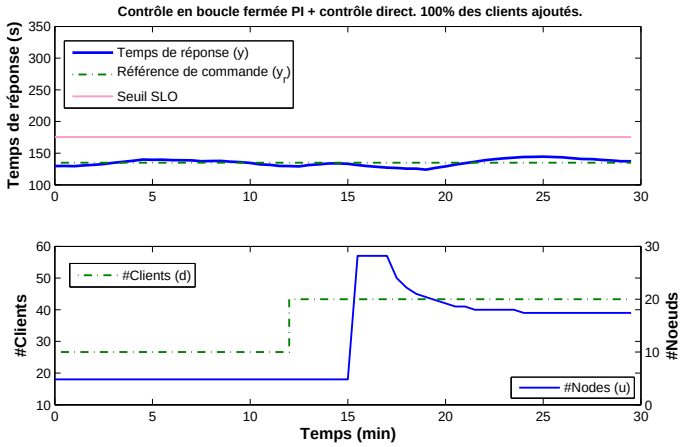

FIGURE 8 – Correcteur PI avec feedforward

bations avant que leurs effets soient observés sur la sortie. Dans notre cas, la perturbation est un changement dans le nombre de clients accédant simultanément à MapReduce. L'idée du correcteur feedforward est de créer un signal de commande qui, une fois qu'il est passé à travers le système, annule l'effet de la perturbation. Plus important encore, ce signal est envoyé en même temps que l'effet de la perturbation se présente. Si le modèle est précis à 100% l'effet sur le temps de réponse doit être nul. Toutefois, en raison des incertitudes inhérentes au modèle, ce n'est jamais le cas dans la pratique.

Le modèle discrétisé du correcteur d'anticipation est donné par $Z_{ff}(z) = -Z_N(z)^{-1}Z_C(z)$ où Z_N e Z_C sont les modèles discrétisés de la Figure 1. La fonction de transfert discrétisée du correcteur d'anticipation Z_{ff} est donnée dans l'équation (5).

$$Z_{ff}(z) = z^{-3}\frac{5.9698(z-0.919)}{(z-0.7915)} \tag{5}$$

En analysant la Figure 8 nous pouvons voir que le correcteur parvient à garder le temps de réponse en dessous du seuil fixé.

4. Etat de l'art

En ce qui concerne la modélisation des performances des tâches MapReduce, l'état de l'art utilise le plus souvent des profils de niveau des *job*. Certains auteurs utilisent des modèles statistiques avec plusieurs invariants de performance tels que la moyenne, le maximum et le minimum du temps d'exécution des différents cycles de MapReduce [22, 27]. Tandis que d'autres emploient un modèle statique linéaire qui prend en compte la relation entre l'exécution du travail, la taille des données d'entrée et le nombre de nœuds de Map, et de Reduce attribués pour la tâche [21]. Dans les deux cas, les paramètres du modèle se trouvent en exécutant la tâche sur un plus petit ensemble de données d'entrée et en utilisant des méthodes de régression linéaire pour déterminer les facteurs d'échelle pour les configurations différentes. Un modèle d'analyse détaillée de la performance hors ligne a été également développée pour l'optimisation des ressources, voir [13]. L'analyse des composants principaux a également été utilisée pour trouver les composants MapReduce/Hadoop qui influencent le plus la performance de MapReduce [28].
Il est important de noter que tous les modèles présentés prédisent la réponse de l'état d'équilibre des tâches MapReduce et ne reflètent pas la dynamique du système. Ils supposent également que un seul travail est en cours d'exécution à un moment dans une grappe, ce qui est loin d'être réaliste. Le modèle de performance que nous proposons répond à ces deux problèmes : il traite une charge de travail issue de plusieurs tâches simultanées et il modélise le comportement dynamique des systèmes MapReduce. En outre, alors que l'approvisionnement des ressources MapReduce pour assurer un accord de niveau de service (SLA)[2] est un domaine de recherche relativement récent dans lequel les efforts sont notables. Certaines approches s'attaquent au problème de trouver la configuration optimale des ressources, au respect des contraintes de temps, par exemple, comme un problème d'optimisation hors ligne, voir [21] et [30]. Cependant, nous pensons que les solutions hors ligne ne sont pas assez solides en situations réelles. Une autre solution est donnée par ARIA, un ordonnanceur capable d'appliquer en ligne les contraintes des délais SLO. Il est construit sur un modèle basé sur les temps des réponse des dernières exécutions. Dans la phase initiale, hors ligne, une quantité de ressources est déterminée et un mécanisme de correction en ligne est déployé. L'entrée de commande est le nombre de slots attribués à une tâche. Il s'agit d'un grave inconvénient car le contrôle ne fonctionne que si la grappe est approvisionné à l'avance avec des places libres pour être consacrer aux tâches. Une autre approche est développée dans Steamengine par [3] qui tente d'éviter l'inconvénient précédent en rajoutant et en supprimant dynamiquement des nœuds dans une grappe.

Cependant, dans tous les cas précédents, il est supposé que chaque travail est en cours d'exécution sur une grappe isolée et par conséquent les exécutions des tâches simultanées, comme dans le cas réel, ne sont pas prises en compte.,

5. Conclusions et Perspectives

Cet article présente la conception, la mise en œuvre et l'évaluation du premier modèle dynamique pour les systèmes MapReduce. En plus, un framework qui permet de garantir le temps de réponse est développé et mis en œuvre avec succès. Nous construisons d'abord, un modèle dynamique

2. SLA est une partie d'un contrat où les contraintes de temps de réponses sont officiellement définies.

boite grise qui saisit avec précision le comportement de MapReduce. Sur cette base nous utilisons une approche théorique de contrôle pour garantir les objectifs de performance. Nous concevons un cadre de contrôle composé d'un correcteur PI classique pour assurer la stabilité et un correcteur par action directe (feedforward) pour améliorer le temps de réponse. L'architecture de contrôle est mise en œuvre dans une grappe avec 40 nœuds en utilisant une charge de travail à forte intensité de données. Nos expériences montrent que la stratégie de contrôle conçue réussi à maintenir les délais fixés préalablement.
En ce moment, les voies de recherches explorées pour améliorer l'approche sont :

1. la mise en œuvre du cadre de contrôle dans un cloud en ligne comme Amazon EC2.
2. mise en place d'une identification en ligne
3. minimisation du nombre de changements de l'entrée de commande à l'aide d'autres techniques de contrôle, tels que la correction à base d'événements [7].
4. tester l'architecture de contrôle utilisant des scénarios de charge de travail plus complexes.

Bibliographie

1. Abdelzaher (T. F.), Shin (K. G.) et Bhatti (N.). – Performance guarantees for web server end-systems : A control-theoretical approach. *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, n1, janvier 2002, pp. 80–96.
2. Cappello (F.), Caron (E.), Dayde (M.), Desprez (F.), Jegou (Y.), Primet (P.), Jeannot (E.), Lanteri (S.), Leduc (J.), Melab (N.), Mornet (G.), Namyst (R.), Quetier (B.) et Richard (O.). – Grid'5000 : A large scale and highly reconfigurable grid experimental testbed. *In : Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pp. 99–106. – Washington, DC, USA, 2005.
3. Cardosa (M.), Narang (P.), Chandra (A.), Pucha (H.) et Singh (A.). – STEAMEngine : Driving MapReduce provisioning in the cloud. *In : 18th International Conference on High Performance Computing (HiPC)*, pp. 1–10. – Bengalore, India, 18-21 Dec. 2011.
4. Chen (Y.), Alspaugh (S.) et Katz (R. H.). – *Design Insights for MapReduce from Diverse Production Workloads*. – Rapport technique nUCB/EECS-2012-17, EECS Department, University of California, Berkeley, Jan 2012.
5. Dean (J.) et Ghemawat (S.). – MapReduce : simplified data processing on large clusters. *Communications of the ACM*, vol. 51, n1, 2008, pp. 107–113.
6. Diao (Y.), Gandhi (N.), Hellerstein (J. L.), Parekh (S. S.) et Tilbury (D. M.). – Using mimo feedback control to enforce policies for interrelated metrics with application to the apache web server. *In : NOMS*, pp. 219–234.
7. Durand (S.) et Marchand (N.). – Further results on event-based pid controller. *In : Proceedings of the European Control Conference (ECC)*.
8. Guillermo J (S.). – *PID Controllers for Time-Delay Systems*. – Birkhauser Boston, 2005.
9. Hellerstein (J. L.), Diao (Y.), Parekh (S.) et Tilbury (D. M.). – *Feedback control of computing systems*. – John Wiley & Sons, Inc., New Jersey, 2004.
10. Herodotou (H.) et Babu (S.). – Profiling, what-if analysis, and cost-based optimization of MapReduce programs. *Proc. of the Very Large Database Endowment (PVLDB)*, vol. 4, n11, 2011, pp. 1111–1122.
11. Kavulya (S.), Tan (J.), Gandhi (R.) et Narasimhan (P.). – An analysis of traces from a production MapReduce cluster. *In : Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 94–103. – Washington, DC, USA, 2010.
12. Kihl (M.), Robertsson (A.) et Wittenmark (B. r.). – Performance modelling and control of server systems using non-linear control theory. *In : Providing quality of service in heterogeneous environments : proceedings of the 18th International Teletraffic Congress, ITC-18, Berlin, Germany, 31 August - 5 September 2003 (Teletraffic science and engineering ; vol. 5)*, éd. par Charzinski (J.), Lehnert (R.) et Tran-Gia (P.). pp. 1151–1160. – Elsevier.

13. Lin (H.), Ma (X.) et Feng (W.-C.). – Reliable MapReduce computing on opportunistic resources. *Cluster Computing*, vol. 15, n2, juin 2012, pp. 145–161.
14. Ljung (L.). – Prediction Error Estimation Methods. *Circuits, systems, and signal processing*, vol. 21, n1, 2002, pp. 11–21.
15. Malrait (L.), Marchand (N.) et Bouchenak (S.). – Modeling and control of server systems : Application to database systems. *In : Proceedings of the European Control Conference (ECC)*, pp. 2960–2965. – Budapest, Hungary, August 23-26 2009.
16. Poussot-Vassal (C.), Tanelli (M.) et Lovera (M.). – Linear parametrically varying MPC for combined quality of service and energy management in web service systems. *In : American Control Conference (ACC), 2010*, pp. 3106–3111. – Baltimore, MD, June 30-July 2 2010.
17. Ren (Z.), Xu (X.), Wan (J.), Shi (W.) et Zhou (M.). – Workload characterization on a production Hadoop cluster : A case study on Taobao. *In : IEEE International Symposium on Workload Characterization (IISWC)*, pp. 3–13. – La Jolla, CA, 4-6 Nov 2012.
18. Rutten (E.), Buisson (J.), Delaval (G.), de Lamotte (F.), Diguet (J.-F.), Marchand (N.) et Simon (D.). – Control of autonomic computing systems. – 2013. Submitted to ACM Computing Surveys.
19. Sangroya (A.), Serrano (D.) et Bouchenak (S.). – Benchmarking Dependability of MapReduce Systems. *In : IEEE 31st Symposium on Reliable Distributed Systems (SRDS)*, pp. 21 – 30. – Irvine, CA, 8-11 Oct. 2012.
20. Tian (C.), Zhou (H.), He (Y.) et Zha (L.). – A dynamic MapReduce scheduler for heterogeneous workloads. *In : Proceedings of the 8th International Conference on Grid and Cooperative Computing (GCC)*, pp. 218–224. – Washington, DC, USA, 27-29 Aug. 2009.
21. Tian (F.) et Chen (K.). – Towards optimal resource provisioning for running MapReduce programs in public clouds. *In : IEEE International Conference on Cloud Computing (CLOUD)*, pp. 155–162. – Washington, DC, USA, 4-9 July 2011.
22. Verma (A.), Cherkasova (L.) et Campbell (R.). – Resource provisioning framework for MapReduce jobs with performance goals. *In : Middleware 2011*, pp. 165–186. – Springer Berlin Heidelberg, 2011.
23. Vernica (R.), Balmin (A.), Beyer (K. S.) et Ercegovac (V.). – Adaptive MapReduce using situation-aware mappers. *In : Proceedings of the 15th International Conference on Extending Database Technology (EDBT)*, pp. 420–431. – New York, NY, USA, 2012.
24. Wang (K.), Lin (X.) et Tang (W.). – Predator ; An experience guided configuration optimizer for Hadoop MapReduce. *In : IEEE 4th International Conference on Cloud Computing Technology and Science*, pp. 419–426. – Taipei, 3-6 Dec. 2012.
25. White (T.). – *Hadoop : the definitive guide*. – O'Reilly Media, CA, 2012.
26. Xie (D.), Hu (Y.) et Kompella (R.). – On the performance projectability of MapReduce. *In : IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 301–308. – Taipei, 3-6 Dec. 2012.
27. Xu (L.). – MapReduce framework optimization via performance modeling. *In : IEEE 26 International Parallel & Distributed Processing Symposium (IPDPS)*, pp. 2506–2509. – Shanghai, China, 21-25 May 2012.
28. Yang (H.), Luan (Z.), Li (W.) et Qian (D.). – MapReduce workload modeling with statistical approach. *Journal of Grid Computing*, vol. 10, 2012, pp. 279–310.
29. Zaharia (M.), Konwinski (A.), Joseph (A. D.), Katz (R.) et Stoica (I.). – Improving MapReduce performance in heterogeneous environments. *In : Proceedings of the 8th USENIX Conference on Operating systems design and implementation (OSDI)*, pp. 29–42. – Berkeley, CA, USA, 8-10 Dec. 2008.
30. Zhang (Z.), Cherkasova (L.), Verma (A.) et Loo (B. T.). – Automated profiling and resource management of pig programs for meeting service level objectives. *In : Proceedings of the 9th International Conference on Autonomic Computing (ICAC)*, pp. 53–62. – San Jose, CA, USA, 17-21 Sept. 2012.